VŠB - Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practise in the Company

2010 Michal Duda

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7. května 2010 …….

podpis

Abstrakt

Práce pojednává o průběhu studentské praxe ve firmě Tieto. Nejdříve firmu Tieto velmi krátce popíši a vysvětlím své umístění a úlohu ve firmě během praxe, představím systém, na jehož vývoji jsem během praxe pracoval. Dále uvádím příklady dvou problémů, na jejichž řešení jsem se podílel. První problém se týká importování a exportování dat ze systému. Druhým úkolem bylo validování vstupů od uživatele do systému. Neopomenu přínos, kterým pro mne praxe byla, zmíním také uplatněné znalosti získané během studia a zároveň scházející dovednosti.

Klíčová slova

.Net, NHibernate, ASP.NET, SQL Server, Tieto, Elevation

Abstract

The topic of this article is my student s practice in Tieto Company. At first I shortly describe Tieto company and my place and task in this company during my practice. I introduce a system on which development I was working during my practice. I also introduce two problems which I took part in solving. The topic of first of them was importing and exporting dates from system. My second task was verifying of user‘s entering to the system. I mention the contribution of the praxis for me, I also mention my knowledge used by study and also missing craft.

Key words

.Net, NHibernate, ASP.NET, SQL Server, Tieto, Elevation

Použité zkratky

XML	eXtensible Markup Language
ASP	Active Server Pages
MVC	Model View Controller
LINQ	Language Integrated Query
IT	Informační technologie
CMS	Content management system
SQL	Structured Query Language
WYSIWYG	What you see is what you get
HTML	HyperText Markup Language

Obsah

1 Popis odborného zaměření firmy

Tieto je nadnárodní společnost poskytující služby v oblasti informačních technologií. Ve firmě Tieto pracuje více než 16 000 odborníků v mnoha různých státech. Tato firma je jedničkou v IT na trhu ve skandinávských zemích, velmi silnou pozici má také například v Německu, Dánsku nebo Rusku.

Tieto poskytuje IT služby mnoha zákazníkům v různých průmyslových odvětvích, například v dřevozpracujícím nebo ropném průmyslu, finančním společnostem a společnostem z odvětví telekomunikací.

2 Popis pracovního zařazení

Ve firmě Tieto jsem byl zařazen na pozici Software Developer, do týmu vyvíjejícího a implementujícího CMS systém Elevation. Tento systém využívá řada velkých společností a také samotné Tieto pro své webové stránky i pro spoustu intranetových projektů. Tým programátorů pracujících na Elevation je rozdělen na část, která se zabývá vývojem systému a na část která se zaměřuje na implementaci konkrétních zákaznických řešení nebo interních projektů. Já jsem byl zařazen do skupiny pro vývoj systému. Celý systém je postavem na technologii .NET a používá databázový server MS SQL Server.

3 Seznam úkolů v průběhu odborné praxe

3.1 Projekt Transfer

3.1.1 Zadání

Transfer je utilita, která je schopna exportovat data ze systému Elevation podle určitých pravidel do formátu XML a také je podle určitých pravidel importovat do jiné databáze. Tato utilita je vytvořena jako konzolová aplikace, která přijímá určité parametry a podle nich provádí import nebo export. Aplikace není určena pro koncové uživatele systému. Je určena pro programátory, kteří připravují konkrétní zákaznická řešení. Na projektu jsem pracoval spolu se třemi dalšími programátory.

Ačkoliv se import a export dat může zdát jako vedlejší a jednodušší úloha, řešení nám zabralo přibližně 3 měsíce.

3.1.2 Řešení

Při řešení jsme použili framework NHibernate [1, 2] a framework S#arp Architecture [3]. NHibernate je framework určený pro objektově relační mapování na platformě .NET. NHibernate provádí mapování podle pravidel, která jsou obvykle definována v XML souborech. Pravidla mohou být také popsána pomocí atributů v jednotlivých doménových objektech. Podle těchto pravidel NHibernate automaticky generuje SQL dotazy pro načítání a ukládání objektů. S#arp Architecture je framework určený k rychlému vývoji webových aplikací založených na ASP.NET MVC a NHibernate. Z tohoto frameworku jsme využili pouze část, která usnadňuje práci s NHibernate, především implementaci návrhového vzoru Repository.

Projekt byl rozdělen do 2 částí:

3.1.2.1 Class Library Transfer.Mappings

Tento projekt obsahuje pouze XML soubory s NHibernate mapováním.

3.1.2.2 Konzolová aplikace Transfer

Tento projekt obsahuje doménové objekty pro NHibernate mapování, třídy pro importování a exportování, třídy pro serializaci a deserializaci doménových objektů z XML a také třídy, které zpracovávají argumenty zadané uživatelem aplikaci a spouští samotný import nebo export.

Projekt Transfer byl velmi rozsáhlý, proto popíši pouze řešení problému serializace doménových objektů a unit testování projektu.

3.1.2.2.1 Serializace doménových objektů do XML

Pro serializaci NHibernate entit není možné použít standardní postup serializace objektů do XML, který poskytuje .NET framework, protože entitní objekty obsahují cyklické závislosti. Byli jsme nuceni napsat vlastní třídy, které pomocí reflexe umožňují provést serializaci nebo deserializaci.

3.1.2.2.2 Řešení cyklických závslostí

K serializaci objektů slouží třída EntitySerializer. Při serializaci objektu jsou procházeny jeho vlastnosti v metodě XmlReferenceType. Metoda využívá rekurze. Jakmile je aktuální zpracovávaná vlastnost referenčního typu, je opět volána tato metoda. Pokud je vlastnost nereferenčního typu

nebo typu String, je do výsledného XML dokumentu přidán element s názvem vlastnosti a její hodnotou. Jelikož entity na sebe navzájem ukazují, je potřeba definovat „hloubku“ zanoření rekurze, aby nedošlo k nekonečnému serializování. Tento problém je řešen pomocí atributu MaxDepth. Tento atribut má jedinou vlastnost depth typu integer. Atribut se připojuje ke vlastnostem referenčních typů jednotlivých entit. Například ve třídě Page, reprezentující entity z tabulky page, nastavíme u vlastnosti PageDataItems hloubku zanoření na hodnotu 3. S tímto se ve třídě EntitySerializer pracuje pomocí reflexe.

```
[MaxDepth(3)]
public virtual IList<PageData> PageDataItems { get; set; }
```

3.1.2.2.3 Ukázka kódu třídy EntitySerializer

Při vytváření třídy je nutné předat entitu, která má být serializována.

```
        public EntitySerializer(object entity)
        {
            _entity = entity;
        }
```

Pro práci s XML jsme v projektu použili LINQ to XML. Celý proces serializace spustí metoda Serialize, která vrátí objekt typu XDocument.

```
        public XDocument Serialize()

        {
                    XElement xe = XmlReferenceType(_entity,
_entity.GetType().Name, 0);
                    XDocument ret = new XDocument(new XDeclaration("1.1",
"utf-8", "no"), xe);
                return ret;
        }
```

Největší část logiky serializace obsahuje privátní metoda XmlReferenceType. Jejím úkolem je serializovat referenční typ entity do objektu XElement. Metoda nejdříve ověří, zda objekt, předaný serializaci, je akceptovaného typu. Dále vytvoří XElement pojmenovaný podle jména typu předané entity. Dále metoda pomocí reflexe prochází jednotlivé vlastnosti objektu a podle jejich typu provádí příslušnou serializaci do XML. Při rekurzivním volání se předává parametr depth označující „hloubku“ rekurze, který je samozřejmě při každém volání navyšován. Pokud je vlastnost označena atributem MaxDepth kontroluje se, zda již nebylo dosaženo maximálního zanoření rekurze.

Ukázka kódu metody (kód je zjednodušený):

```
private XElement XmlReferenceType(object obj, string elementName, int
depth)
        {
            if (!IsAcceptedType(obj))
            {
                  return null;
            }

            Type type = obj.GetType();
            XElement ret = new XElement(elementName);
            PropertyInfo[] pi = type.GetProperties();
```

```
            foreach (PropertyInfo info in pi)
            {
                  if (info.Name == "Id")
                        continue;
                  object[] attribs =
info.GetCustomAttributes(typeof(XmlIgnoreAttribute), false);
                  if (attribs.Length > 0)
                        continue;

                  Type depthType = typeof(MaxDepthAttribute);
                  attribs = info.GetCustomAttributes(depthType, false);
                  if (attribs.Length > 0)
                  {
                  // Nalezen atribut určující maximální zanoření,
                  // v tomto bloku se provede kontrola, zda již nemá
                  // být serializování ukončeno
                  }

// Rozeznávání typu vlastnosti, provedení serializace podle nalezeného
typu

                  Type collection =
info.PropertyType.GetInterface("ICollection`1");

                  if (collection != null)
                  {
                  // Provedení serializace kolekce
                  }
                  else if  (IsNativeType(info.PropertyType.FullName) ||
info.PropertyType == typeof(string))
                  {
                  // Uložení jednoduchého nereferenčního typu nebo
řetězce
                  }
                  else {
                  // Vlastnost je referenčního typu, provede se
                  rekurzivní // volání metody pro serializaci
                  referenčního typu
                  XElement xe = XmlReferenceType(info.GetValue(obj,
null), info.Name, depth + 1);
                  ret.Add(xe);
                 }
            }

            return ret;
        }
```

3.1.3 Unit testování projektu Transfer

Projekt Transfer byl velmi složitý a řešil velmi velké množství různých případů exportování a importování dat, podle zadaných pravidel. Proto vznikly dva projekty pro testování. Pro testování je použit framework NUnit [4] (NUnit je framework pro snadné vytváření unit testů ve všech jazycích platformy .NET).

3.1.3.1 Projekt Transfer.Tests

Tento projekt obsahuje unit testy pro všechny třídy projektu transfer, tedy pro serializaci, deserializaci, export i import entit. Testy pro export a import se provádí nad databází SQLite,

kterou NHibernate automaticky vytvoří podle aktuálního schématu entit. Všechny třídy s unit testy, které pracují s SQLite databází dědí z bázové třídy SQLiteTestBase.

Ukázka z konfigurace NHibernate pro vytvoření SQLite databáze:

```
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
	<session-factory>
	<!-- Nastavení driveru pro SQLite -->
	<property
name="connection.driver_class">NHibernate.Driver.SQLite20Driver</property
>

	<!– Nastavení property hbm2ddl.auto na create zajistí vytvoření
databáze -->
	<property name="hbm2ddl.auto">create</property>

	<!– Nastavení assembly, která obsahuje NHibernate mapování -->
	<mapping assembly="Transfer.Mappings"/>
	…

	…

	… Další NHibernate nastavení

	</session-factory>
</hibernate-configuration>
```

Ukázka kódu bázové třídy pro Unit testy:

Vytvoření NHibernate Configuration před spuštěním každého testu:

```
[SetUp]
public void SetUp()
{
	_config = NHibernateSession.Init(new SimpleSessionStorage(), new
string[] { }, "sqlite.cfg.xml");
}
```

Vyčištění NHibernate cache a smazání SQLite databáze po každém testu:

```
[TearDown]
public void TestFixtureTearDown()
{
	DeleteTestDb();
	NHibernateSession.SessionFactory.Dispose();
	NHibernateSession.SessionFactories.Clear();
	NHibernateSession.Storages.Clear();
}

private void DeleteTestDb()
{
	string filename = "SQLite.Test.db";
	if (File.Exists(filename))
	{
		File.Delete(filename);
	}
}
```

3.1.3.1.1 Projekt Transfer.Import.Big.Tests

Tento projekt je sadou integračních testů. Projekt tzv. velkých testů importu dat nepracuje s SQLite databází nýbrž vytváří databázi v MS SQL serveru. Databáze se vytvoří pomocí skriptu, který se používá při instalaci systému Elevation. Skript se jmenuje SQLRunner, načítá postupně přiložené soubory s SQL skripty a vytváří datovou strukturu pro systém Elevation. Po provedení tohoto kroku je několikrát spuštěna aplikace Transfer nad připravenými různými exporty dat. Jakmile program Transfer skončí, jsou postupně spouštěny metody, které načítají data z databáze a kontrolují, zda import proběhl správně. Každá metoda testuje proces načtení určitého XML souboru, deserializaci jeho obsahu a uložení pomocí některé třídy pro import. Oproti prvnímu projektu pro testování se provádí testy větších celků z projektu transfer.

Ukázka vytvoření MS SQL databáze:

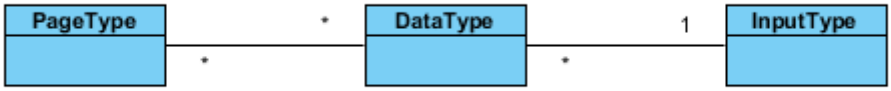
```
public static void CreateDatabase()
{
	/// Vytvoření procesu pro spuštění skriptu na vytvoření databáze
	Process dbScript = new Process();
	/// Potřebné parametry jsou načítány z konfiguračního souboru unit
	///  testů, například umístění skriptu, databázový server atd.
	dbScript.StartInfo.Arguments = string.Format(
		@"/c	cscript	""{0}""	/source:""{1}""	/server:{2}
		/database:{3} /create",
		ConfigurationManager.AppSettings["SqlRunnerPath"],
		ConfigurationManager.AppSettings["SqlRunnerSource"],
		ConfigurationManager.AppSettings["SqlRunnerServer"],
		ConfigurationManager.AppSettings["SqlRunnerDatabase"]);

	dbScript.StartInfo.FileName = "cmd.exe";
	dbScript.StartInfo.UseShellExecute = false;
	dbScript.StartInfo.CreateNoWindow = true;
	dbScript.StartInfo.RedirectStandardOutput = true;
	dbScript.Start();

	/// Před dalším pokračováním testů je nutné počkat na dokončení ///
	/// skriptu
	dbScript.WaitForExit();
	Trace.Write(dbScript.StandardOutput.ReadToEnd());
}

[TestFixtureSetUp]
public void TestFixtureSetUp()
{
	CreateDatabase();

	/// Spuštění programu Transfer s příslušnými parametry pro import
	CommandLineArguments args = new CommandLineArguments(new string[] {
				"/target:" + _connString,
				@"/folder:FirstImportData",
				"/levels:0",
				"/targetpage:" + _secondTargetPath,
				"/sourcepage:1;38",
				"/keeproot",
				"/targetlogin:admin" });
	Program p = new Program(args);
	p.Run();

}
```

4 Validace uživatelských vstupů v administrátorské části

4.1.1 Zadání

Administrátoři, manažeři a redaktoři vytvářejí obsah internetové prezentace pomocí internetové aplikace nazvané Management site. Přistup do této části systému mají samozřejmě pouze někteří uživatelé. Uživatelé, kteří pracují s Management site mají různá práva a jsou zařazeni do různých skupin. Uživatelé s nejvyššími právy mohou vytvářet takzvané typy stránek (page type). Každý typ stránky specifikuje z jakých datových typů (data type) se bude stránka skládat. Obvyklým příkladem může být typ, který specifikuje, že stránka bude obsahovat datový typ Title, datový typ ShortDescription a datový typ TextContent. Každý datový typ má dále nastaven vstupní typ (input type). Každý vstupní typ představuje část uživatelského rozhraní pro zadávání dat, tedy například vstupní typ „SingleLine“ představuje obyčejný řádek pro zadání krátkého textu, vstupní typ „HTMLEdit“ představuje webový WYSIWYG editor TinyMCE [5]. Všechny vstupní typy jsou při vytváření nové stránky uživateli vyrenderovány pomocí komponenty PageEditForm.

Propojení tříd v systému bude lépe pochopitelné z class diagramu, viz Obrázek 1:



Obrázek 1: Propojení tříd PageType, DataType, InputType

Každý typ stránky může obsahovat více datových typů, stejně tak datový typ může být použit ve více typech stránky. Každému datovému typu je přiřazen jeden vstupní typ.

Mým úkolem bylo umožnit validaci uživatelských vstupů v administrátorské části systému Elevation. Bylo třeba nejen naimplementovat některá klasická pravidla pro validaci, jako je například maximální délka textu nebo validní tvar emailové adresy, ale také vytvořit rozhraní, které umožní ostatním programátorům vytvářet vlastní třídy obsahující validační pravidla. Jednotlivá pravidla poté musí být možno připojit ke vstupním typům v systému a určit, zda se pravidlo vztahuje obecně ke vstupnímu typu, datovému typu nebo typu stránky. Bylo tedy zapotřebí vytvořit uživatelské rozhraní, které umožní zvolit určité pravidlo, přiřadit jej do některé ze tří oblastí působnosti (vstupní typ, datový typ, typ stránky) a přiřadit mu požadovanou hodnotu. Tato data samozřejmě musí být uložena, proto bylo potřeba vytvořit novou tabulku v databázi a vytvořit API pro práci s touto tabulkou.

4.1.2 Řešení problému

4.1.2.1 Rozhraní pro programátory implementující vlastní validace

Programátoři mají možnost vytvářet vlastní třídy, které obsahují metody pro validaci. Každá metoda, určená k validaci musí být označena speciálním atributem, například:

```
[InputHandlerConfiguration("Maximum length", "150")]

public virtual void ValidateMaxLength(string value, string
configurationValue)
{

      int length = int.Parse(configurationValue);

      if (value.Length > length)
            throw new ValidationException(string.Format("Maximal allowed
            length of text is {0}.", length),
            "EMS:InputHandlers.Validation.MaximumLength", new object[] {
            length });
      }

}
```

Atribut InputHandlerConfiguration specifikuje především název validátoru, který je použit i v uživatelském rozhraní. Dále může volitelně specifikovat výchozí hodnotu a určovat, zda uživatel může zadat hodnotu pro validaci. Například při použití validátoru pro emailovou adresu nechcete, aby uživatel zadával hodnotu, regulární výraz pro validaci emailu je vždy stejný.

Metody dále musí přijímat dva parametry typu string, první je hodnota, kterou uživatel do vstupního pole zadal a druhá je hodnota, která se má použít pro validování. Může jí být například maximální délka, kterou má hodnota mít.

Programátoři, kteří implementují zákaznická řešení, musí nějakým způsobem aplikaci Management site sdělit, kde najde jimi implementované validační třídy a pro které vstupní typy mohou být použity. Toto nastavení se provádí v souboru web.config.

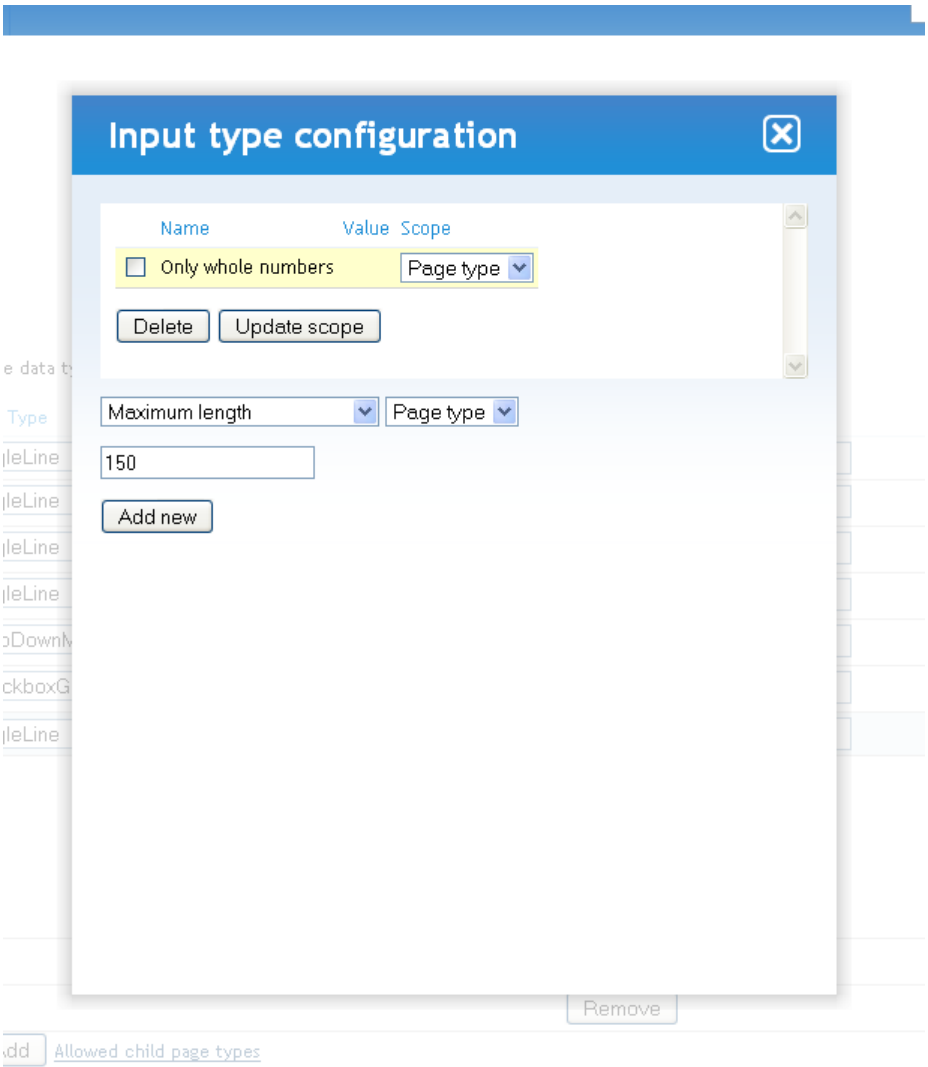
Ukázka nastavení v souboru web.config:

```
<inputHandlers>
      <add type="TietoEnator.Elevation.Web.UI.InputHandlers.SingleLine,
DynamitePlus.NET" inputTypes="SingleLine AreaDimensions">
            <validator
type="TietoEnator.Elevation.Web.UI.InputHandlers.Validators.BaseValidator
, DynamitePlus.NET" />
      </add>
      <add type="TietoEnator.Elevation.Web.UI.InputHandlers.TextArea,
DynamitePlus.NET" inputTypes="SmallTextArea BigTextArea" />
      …

      …

      … seznam dalších vstupních typů

</inputHandlers>
```

V sekci inputHandlers jsou uvedeny vstupní typy. V elementu add je atribut type, který obsahuje plný název třídy a název assembly oddělené čárkou. Každý element add může obsahovat elementy validator, které stejným způsobem specifikují třídu obsahující validační metody. Typy těchto validačních tříd jsou načteny při prvním startu aplikace a jsou uloženy v objektech jednotlivých vstupních typů jakožto kolekce objektů typu Type. Instance validačních objektů se vytvářejí teprve až je to zapotřebí.

4.1.2.2 Uživatelské rozhraní pro ukládání validačních pravidel

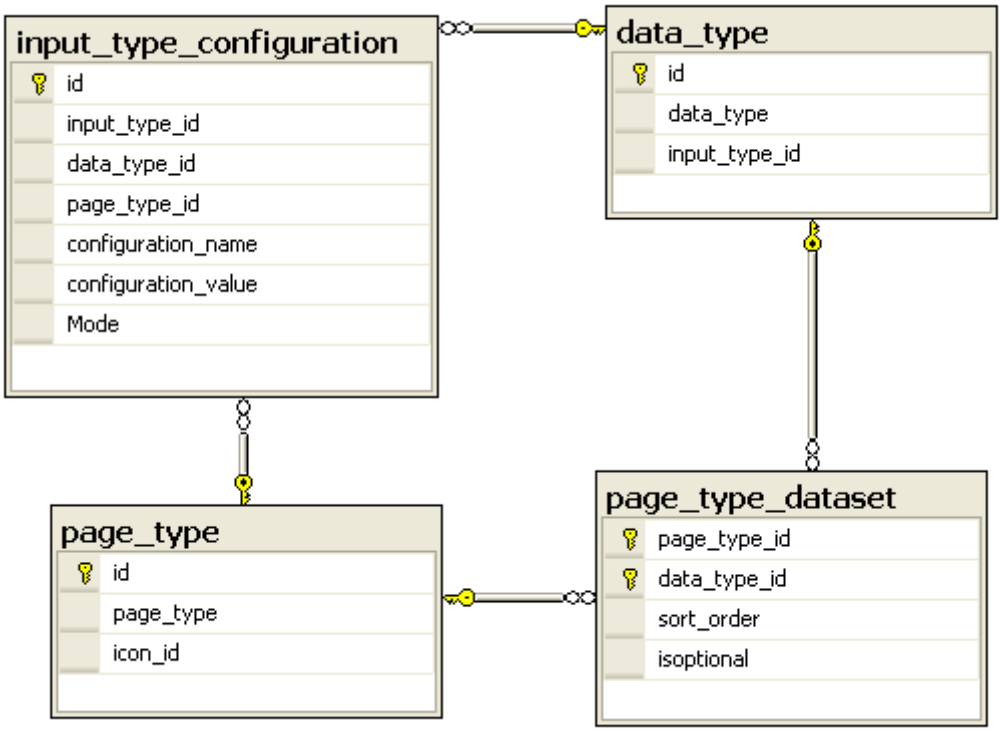
Uživatelským rozhraním pro ukládání validačních pravidel je ASP.NET webová stránka. Uživatel nejdříve v jiné části Management site aplikace vybere, se kterým typem stránky, datovým typem a vstupním typem chce pracovat a poté je mu zobrazen seznam aktuálních validačních pravidel s možností editace a přidávání nových pravidel. **Pomocí tohoto uživatelského rozhraní tedy administrátor webu určuje, která z možných validačních pravidel se budou při ukládání dat aplikovat na určitý typ stránky, datový typ a vstupní typ.** Rozhraní je vytvořeno jako dialogové okno v aplikaci ve webové aplikaci (viz Obrázek 2).



Obrázek 2: Uživatelské rozhraní pro správu validačních pravidel nad určitým typem stránky, datovým typem a vstupním typem

4.1.2.3 Ukládání zvolených validátorů

Uložení se provádí do tabulky, která obsahuje název validátoru a hodnotu, podle které se má validace provést. Oba sloupce jsou typu varchar. Kromě těchto sloupců obsahuje tabulka ještě tři cizí klíče – cizí klíč z tabulky vstupních typů, cizí klíč z tabulky datových typů a cizí klíč z typů stránek (propojení tabulek viz Obrázek 3). Zadán musí být pouze cizí klíč z tabulky vstupních typů.

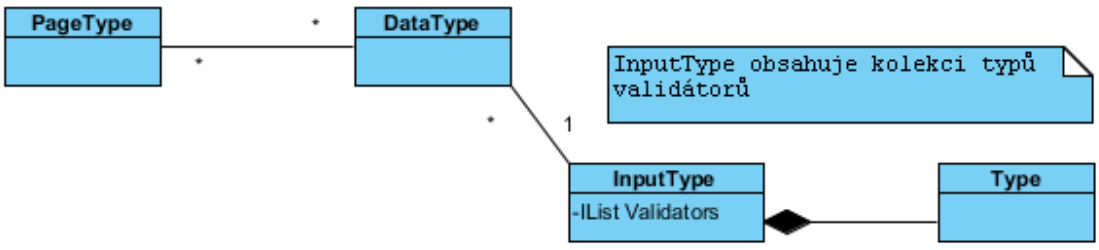


Obrázek 3: Tabulka input_type_configuration a napojení na ostatní tabulky v systému

Komunikace s databází je v systému Elevation vždy prováděna pomocí uložených procedur. Proto bylo také potřeba vytvořit procedury pro načítání, ukládání a mazání pravidel.

4.1.2.4 Provedení validace při ukládání nové stránky do systému

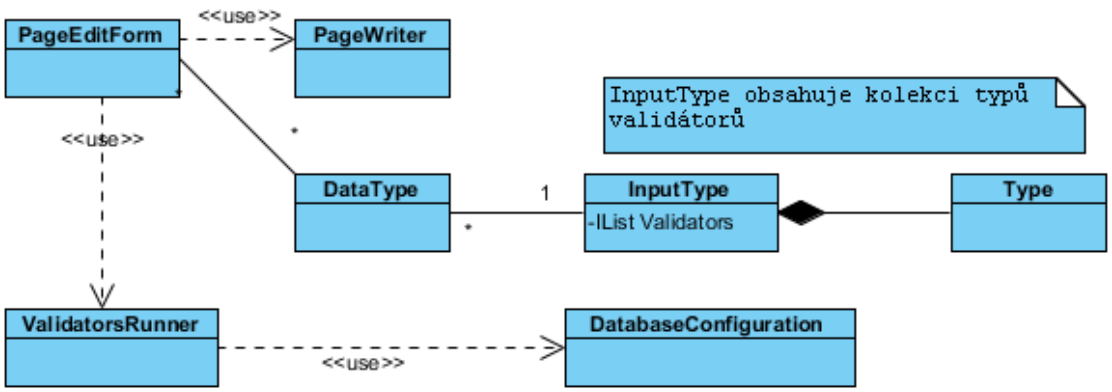
Formulář pro vytvoření nebo editaci stránky vytváří komponenta PageFormEdit. Komponenta obsahuje kolekci datových typů stránky, datové typy zase obsahují odkaz na objekt svého vstupního typu (propojení tříd viz Obrázek 4).



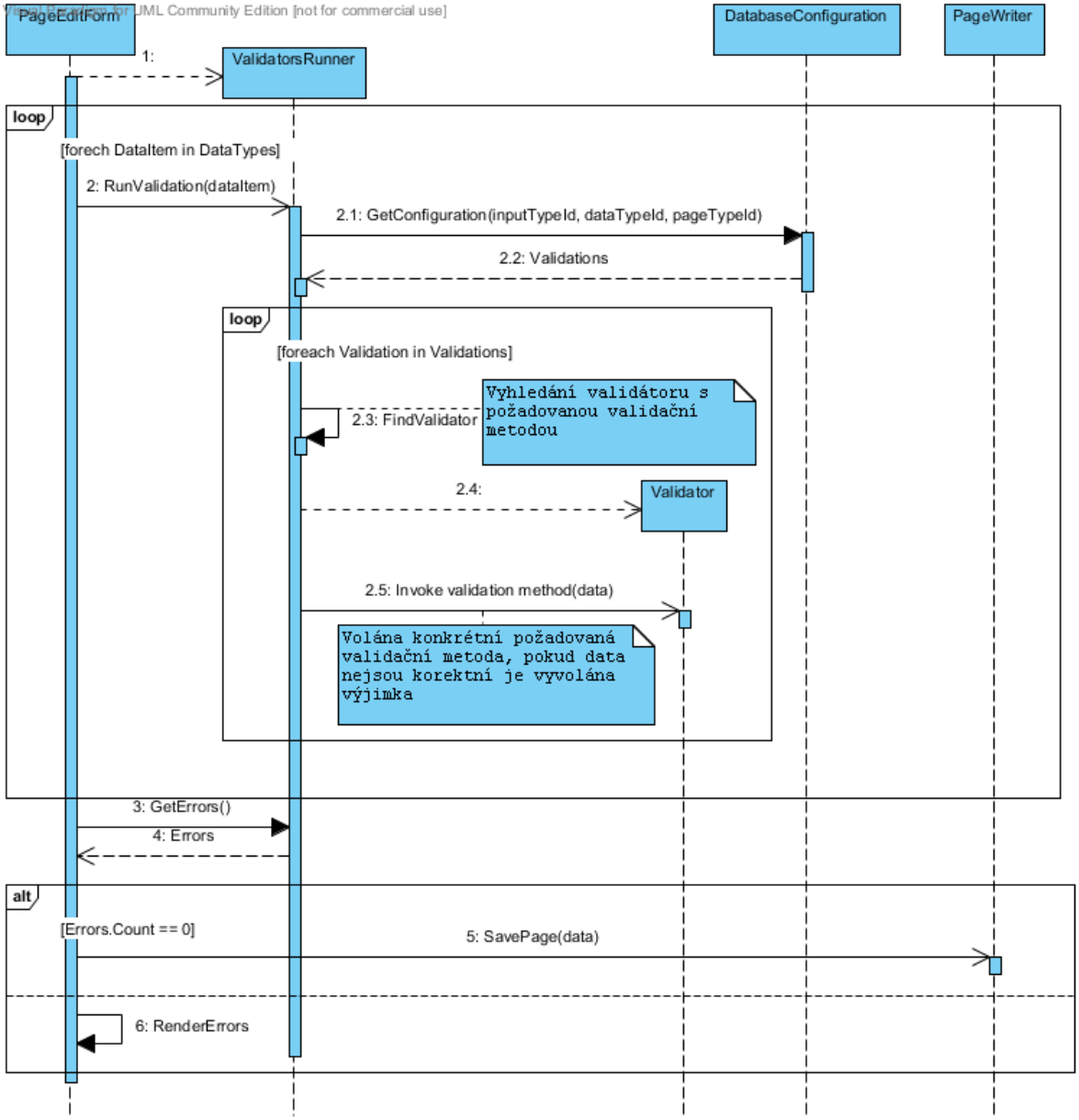
Obrázek 4: Třída InputType obsahuje kolekci typů validátorů

Komponenta PageFormEdit se postará o vyrenderování jednotlivých vstupních polí a v případně nekoretních vstupů také o vypsání chybových hlášení. Pokud jsou zadaná data v pořádku, předá je datové vrstvě pro uložení.

Pro spouštění validací je vytvořen objekt ValidatorsRunner podle stejnojmenné třídy. Komponenta PageEditForm tomuto objektu předá id aktuálního typu stránky, datového typu i vstupního typu. ValidatorsRunner načte aktuální nastavení pro tyto hodnoty z databáze. Pro načítání veškerých konfigurací a nastavení se v systému Elevation používá třída DatabaseConfiguration. Komponenta PageEditForm postupně spouští metodu RunValidation objektu ValidatorsRunner na všech datových položkách vyplněných uživatelem. V metodě RunValidation jsou procházeny záznamy z databáze a podle nich vyhledávány metody pro validaci. Nalezená metoda pro validaci je následně spuštěna. Pokud některá z validací selže, je vyvolána speciální výjimka ValidationException. Tuto výjimku odchytí třída ValidatorsRunner a uloží do kolekce všech nalezených chyb. Kolekci chyb si nakonec vyžádá komponenta PageEditForm. Pokud kolekce není prázdná, zabrání komponenta PageEditForm uložení stránky. Také zajistí zobrazení příslušného chybového hlášení pro uživatele. Pokud nebyla zaznamenána žádná chyba, provede se uložení stránky (propojení tříd viz Obrázek 5, průběh validace viz Obrázek 6). Pro ukládání stránek se v systému Elevation používá třída PageWriter.



Obrázek 5: Kompletní class diagram tříd spolupracujících při ukládání stránky v systému



Obrázek 6: Sekvenční diagram znázorňující provedení validace na určitém datovém typu

5 Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe

Během praxe jsem nejvíce uplatnil znalosti získané v těchto předmětech:

1. Programování v C#
2. Architektura .NET
3. Databázové a informační systémy
4. Tvorba informačních systémů
5. Teorie zpracování dat
6. Vývoj internetových aplikací

6 Znalosti či dovednosti scházející studentovi v průběhu odborné praxe

Nejvíce mi scházela hlubší znalost technologie .NET, především ASP.NET. Také mi chyběla znalost technologie NHibernate, znalost práce s databází MS SQL.

Vůbec jsem neznal nástroje pro týmovou práci od společnosti Microsoft – Team Foundation Server.

Téměř žádné znalosti jsem neměl také v agilních metodikách vývoje software.

7 Dosažené výsledky a jejich celkové zhodnocení

Praxe ve firmě Tieto byla pro mne obrovská zkušenost. Zjistil jsem, jak vypadá práce na skutečných projektech a jak se pracuje v týmu několika lidí. Velmi jsem si prohloubil znalosti z technologie .NET, naučil jsem se pracovat s programy pro vývoj a týmovou spolupráci od firmy Microsoft.

Kromě zkušeností jsem díky praxi získal také pracovní úvazek. Po několika měsících praxe mi bylo umožněno změnit smlouvu s firmou Tieto z dohody o provedení práce, kterou firma normálně studentům na praxi poskytuje, na pracovní smlouvu na dobu neurčitou. U firmy Tieto v týmu Elevation tedy zůstávám na částečný úvazek i nadále po skončení praxe.

8 Použitá literatura

[1] Pierre Henri Kuaté, Tobin Harris, Christian Bauer, and Gavin King, *NHibernate in Action.* 1. vyd. Manning Publications Co., 2009. 400 s. ISBN: 1932394923

[2] *NHibernate Forge.* URL: <http://nhforge.org> [cit. 22.4.2010].

[3] *Sharp Architecture*. URL: < http://wiki.sharparchitecture.net> [cit. 22.4.2010].

[4] *NUnit documentation.* URL: <http://www.nunit.org/index.php?p=documentation> [cit 22.4.2010].

[5] *TinyMCE - Javascript WYSIWYG Editor* URL: < http://tinymce.moxiecode.com/> [cit 22.4.2010]